ABSTRACT

Neural networks are often selected as the tool for software effort prediction because of their capability to approximate any continuous function with arbitrary accuracy. A major drawback of neural networks is the complex mapping between inputs and output which is not easily understood by a user. This paper describes a rule extraction technique that derives a set of comprehensible IF-THEN rules from a trained neural network applied to the domain of software effort prediction. The applicability of this technique is tested on the ISBSG R11 data set by a comparison with linear regression and CART. It is found that the most accurate results are obtained by CART, though the large number of rules limits comprehensibility. Considering comprehensible models only, the concise set of extracted rules outperform the pruned CART rule set, making neural network rule extraction the most suitable technique for software effort prediction.

Categories and Subject Descriptors

D.2.9 [Cost Estimation]: [Rule extraction for Artificial Neural Networks in software effort prediction]

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

ANTWERP 2010, Belgium Europe
Copyright 2010 ACM X-XXXX-XX-X/XX/XX ..$10.00.

General Terms

Software engineering

Keywords

Data mining, Software effort prediction, Neural Networks

1. INTRODUCTION

Resource planning is considered a key issue in business environments. In the context of a software developing company, the different resources are amongst others computers, workspace and personnel. In recent years, computing power has become a less important resource for software developing companies [46]. Hence proper planning of personnel effort is a key aspect for companies. There is a growing interest in the literature involving software effort prediction [24]. In this field of research, the effort needed to develop a new project is estimated based on historical data from previous projects. This information can be used by the management to improve the planning of personnel, to make more accurate tendering bids and to evaluate risk factors.

The first attempts to estimate software development effort date back to the late 60’s [33]. In these cases, expert judgement in which a domain expert applies his/her prior experience to estimate the effort was utilized. Since then, a myriad of estimation techniques have been applied to software effort prediction [10].

Most of the software effort estimation models are either formal models or data mining approaches. Some of the most frequently used models include COCOMO, SLIM and FPA. Constructive Cost Model (COCOMO) is a formal model published in 1981 by B. Boehm. The original COCOMO I model related only project size to development effort. A
commonly used derivative model is the COCOMO I intermediate model which has the following form

\[
\text{Effort} = a \cdot \text{size}^b \cdot \left( \prod_{j=1}^{15} E M_j^{a_j} \right)
\]

where \( E M_1 \ldots E M_{15} \) are effort multipliers which are used to take specific properties of a project into account. Size is expressed in SLOC (Source Lines Of Code) or an equivalent measure and \( a \), \( b \) and \( a_j \) are calibration coefficients [9]. The calibration coefficients can be estimated using statistical techniques (e.g. regression) or taken from the literature.

Another formal model proposed by Putnam is SLIM (Software Life cycle Model) which also depends on SLOC to estimate the size of a project [36]. This size estimation is then modified by using the Rayleigh curve model to provide a software size estimate. The shape of the Rayleigh curve can be tuned by two parameters, a ‘manpower buildup index’ which determines the slope of the curve and a productivity factor.

Not all formal models are based on SLOC as input. For instance Function Point Analysis (FPA) is a formal model in which the effort is determined by function points as a measure for project size [2]. In FPA, an unadjusted function point count (UFP) is calculated by taking the number of user functions into account. The number of function points (FP) is determined by the combination of the unadjusted function point count and the Value Adjustment Factor (VAF). The VAF is calculated by taking a number of General System Characteristics, \( GSC_1 \ldots GSC_{14} \), into consideration.

\[
\text{FP} = \text{UFP} \cdot \left( 0.65 + 0.01 \sum_{j=1}^{14} GSC_j \right)
\]

Based on a comparison of the function points of previous projects, the software effort can be estimated.

An advantage of formal models is their comprehensibility; the relation between the various aspects of the model and the effect on the resulting effort prediction is captured by a single mathematical formula that monotonically relates the software development characteristics to the development effort. Since 1990 however, research interest in these formal models has declined considerably [24]. Data mining approaches which are able to generalize after being trained on a data set have been the focus of different studies. Data mining approaches include techniques like regression techniques, Classification And Regression Trees, Neural Networks, Case Based Reasoning and others [45]. Some of these techniques, e.g. Neural Networks, can be considered to be black boxes as the relation between inputs and output is non linear thus limiting the comprehensibility to the end user [25] [40]. Due to a lack of comprehensibility, the applicability of Neural Networks in business environments is limited.

Data sets in this field tend to be difficult to collect due to the nature of the data. Table 1 provides a non exhaustive overview of a number of data sets frequently used in the software effort prediction literature. The ISBSG R11 data set is one of the largest data sets for software effort prediction. A number of previous studies assessed the applicability of NNs in software effort estimation. For instance, a study performed by Srinivasan et al. compared CART and NNs using a combination of the COCOMO81 and the Kemerer data set. By training a backpropagation network with 5 hidden neurons, a MMRE (Mean Magnitude of Relative Error, defined in Section 4.2) of 104 % was obtained. It was concluded that CART and NNs were on par with formal model approaches [44]. Finnie et al. compared NNs, regres-

<table>
<thead>
<tr>
<th>Data set</th>
<th>Number of projects</th>
<th>Number of features</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kemerer [26]</td>
<td>15 projects</td>
<td>3 feat.</td>
</tr>
<tr>
<td>Mendes et al. 2 [32]</td>
<td>37 projects</td>
<td>28 feat.</td>
</tr>
<tr>
<td>Walston-Felix [50]</td>
<td>60 projects</td>
<td>29 feat.</td>
</tr>
<tr>
<td>Mendes et al. 1 [31]</td>
<td>76 projects</td>
<td>6 feat.</td>
</tr>
<tr>
<td>Experience</td>
<td>856 projects</td>
<td>17 feat.</td>
</tr>
<tr>
<td>ISBSG R11</td>
<td>5052 projects</td>
<td>58 feat.</td>
</tr>
</tbody>
</table>
sion and Case Based Reasoning on the ASMA data set. A MMRE of 352% was obtained. Comparing the results obtained by the other techniques, the authors concluded that regression was unable to correctly capture the complexities in the data set while artificial intelligence models were capable to provide adequate estimation models [17]. A study by Wittig et al. assessed NNs using the Desharnais data set and an artificially generated data set. A MMRE of 17% was found and thus it was concluded that NNs can be used in a software effort prediction context [52]. Heiat assessed two different types of NNs, multilayered perceptron and radial basis function networks using an IBM data set and the Kemerer data set. Depending on the data set, a MMRE of 47.60% and 90.27% was obtained for the MLP NN. The author concluded that the NN approach was competitive to the regression approach [18].

It can be concluded from previous research that NNs can be applied with success to software effort prediction. However, as the mapping from inputs to output in a neural network is unclear to the end user, the utility in a business context is limited. Comprehensibility is of key importance as the predictive model needs to be validated before use in a business context [49] [34] [30]. Well documented formal models like COCOMO and FPA are often preferred in businesses for this reason. Also techniques from which a rule set can be derived can be used (e.g. CART) to this end.

A number of rule extraction techniques exists that derive a comprehensible rule set from a NN. The majority of these rule induction techniques is only applicable in the context of classification problems (i.e. with a discrete class variable) [4]. In case of a continuous target attribute (e.g. effort), one possibility is to discretize the target using equal width, equal binning or clustering and then apply rule extraction techniques. This idea is used for instance in the RECLA system [47]. In this paper a method to generate linear regression rules from trained neural networks for software effort prediction is applied. Unlike RECLA, the proposed method does not require discretization of the effort.

3. TECHNIQUES

In this section, the regression rule extraction from NNs is described. First, an introduction to CART, OLS and NNs is provided.

The following notation is used throughout the remainder of the paper. A scalar $x \in \mathbb{R}$ is denoted in normal script while a vector $x \in \mathbb{R}^n$ is in boldface script. A vector is always a column vector unless indicated else. A row vector is indicated as the transposed of the associated column vector, $x'$. A matrix $X \in \mathbb{R}^{N \times n}$ is in bold capital notation. $x_i(j)$ is an element of matrix $X$ representing the value of the $j^{th}$ variable on the $i^{th}$ observation. $N$ is used as the number of observations in a data set while $n$ represents the number of variables.

In the ISBSG data set, the target variable used is effort in man-hours. The actual effort of the $i^{th}$ software project is indicated as $e_i$ while the predicted effort is indicated as $\hat{e}_i$ in line with the previous introduced notation.

### 3.1 Linear regression

Arguably, one of the oldest and most widely applied techniques for software effort estimation is OLS regression. In case of this well documented technique, the goal is to fit a linear regression function to a data set containing a dependent, $e_i$, and multiple explanatory variables, $x_i(1)$ to $x_i(n)$; this type of regression is also commonly referred to as multiple regression. OLS regression assumes the following linear model of the data,

$$e_i = x_i' \beta + \beta_0 + \epsilon_i$$

where $x_i'$ represents the row vector containing the values of the $i^{th}$ observation, $x_i(1)$ to $x_i(n)$. $\beta$ is the column vector containing the slope parameters that are to be estimated and $\beta_0$ is the intercept scalar. This intercept can also be included in the $\beta$ vector by introducing an extra variable with value ‘1’ for each observation. $\epsilon_i$ is the error associated with each observation.

This error term is used to estimate the regression parameters, $\beta$, by minimizing the following objective function:

$$\min_{\beta} \sum_{i=1}^{N} \epsilon_i^2$$

thus obtaining the following estimate for $\beta$:

$$\hat{\beta} = (X'X)^{-1}X'e$$

with $e$ representing the column vector containing the effort and $X$ a $N \times n$ matrix with the associated explanatory variables.

### 3.2 CART

CART (Classification And Regression Tree) is an algorithm that takes the well known idea of decision trees for classification [37] and adjusts it to a regression context. CART constructs a binary tree by recursively splitting the data set until a stopping criterion has been met [11]. The splitting criterion used in this study is the least squared deviation:

$$\min \left[ \sum_{i \in L} (e_i - \bar{e}_L)^2 + \sum_{i \in R} (e_i - \bar{e}_R)^2 \right]$$

The data set is split in a left node (L) and a right node (R) in such a way that the sum of the squared differences between the observed and the average value is minimal. The stopping criterion is set as a minimum of 10 observations at the terminal nodes. Fig. 1 presents a binary regression tree constructed by applying CART to the ISBSG R11 data set. The good comprehensibility of regression trees can be considered a strong point of this technique. To determine the effort needed for a new project, it is sufficient to select the appropriate branches based on the characteristics of the new project. It is possible to construct an equivalent rule set based on the obtained regression tree (Fig. 1, bottom).

This technique has previously been applied within a software effort prediction context where it consistently was considered to be one of the better performing techniques [13] [12] [27] [23].

### 3.3 Neural networks

Neural networks (NNs) are mathematical representations inspired by the functioning of the human brain. Many types of neural networks have been suggested in the literature [7]. In the field of software effort prediction, multilayer perceptron (MLP) networks are the most common used type of
neural networks. A MLP is typically composed of an input layer, one or more hidden layers and an output layer, each consisting of several neurons. With the inputs of each of the neurons, a weight is associated. Assume \( w_m \) is the vector of weights associated with the inputs of the \( m \)th hidden unit and \( v_m \) is the scalar representing the weight associated with the output of the \( m \)th hidden unit.

The output of the hidden units are calculated by a hyperbolic tangent transfer function, \( \tanh(\xi) \).

\[
\tanh(\xi) = \frac{e^\xi - e^{-\xi}}{e^\xi + e^{-\xi}}
\]

The final effort prediction, \( \hat{e} \), is formed by taking the sum of the outputs of the hidden units and correcting for the output bias \( \tau \). Other transfer functions are possible. Fig. 2 illustrates the principle of a feedforward NN.

\[
\hat{e}_i = \sum_{m=1}^{H} \tanh(x'_i w_m) v_m + \tau
\]

The network is trained by minimizing the sum of squared errors \( E(w, v) \) augmented with a complexity penalizing term \( P(w, v) \):

\[
E(w, v) = \sum_{i=1}^{N} (\hat{e}_i - e_i)^2 + P(w, v)
\]

Once the network has been trained, irrelevant and redundant hidden units and input units are removed from the network by applying the algorithm N2PFA (Neural Network Pruning for Function Approximation) [41].

### 3.4 Regression rule extraction

A set of regression rules can be induced from the trained NN by applying REFANN (Rule Extraction from Function Approximating Neural Networks) [39] [42]. This technique uses a piece-wise linear approximation, \( L(\xi) \), of the hyperbolic tangent activation function, \( \tanh(\xi) \), consisting of three line segments. Fig. 3.

The 3-piece linear approximation, \( L(\xi) \), is obtained by minimizing the sum of the squared deviations:

\[
\min_{\xi_0, \alpha_1, \beta_1} \sum_{i=1}^{K} (\tanh(\xi_i) - L(\xi_i))^2,
\]

where \( \xi_i = v'_i w \), the weighted input of sample \( i \), \( i = 1, 2, \ldots N \) and

\[
L(\xi) = \begin{cases} 
-\alpha_1 + \beta_1 \xi & \text{if } \xi < -\xi_0 \\
\beta_0 \xi & \text{if } -\xi_0 \leq \xi \leq \xi_0 \\
\alpha_1 + \beta_1 \xi & \text{if } \xi > \xi_0
\end{cases}
\]

As \( \tanh(0) = 0 \), \( L(0) \) is also fixed at 0. Due to symmetry of the \( \tanh(\xi) \) function, the slope of the first and third line segment are equal and the intercept between the first (third) and the middle line segment is \( -\xi(\xi) \).

A set of comprehensible IF-THEN regression rules can be extracted as the hyperbolic tangent transfer function for each hidden neuron \( m = 1 \ldots H \) is approximated by a 3-piece linear function. The procedure for rule extraction is as follows.

- The input space is divided into \( 3^H \) subregions by using the pair of intercepts (\( \xi \) and \( -\xi \)) from the function \( L_m(\xi) \).
- For each non-empty subregion, a rule is generated as follows:

1. Define a linear equation that approximates the network’s output for input sample \( i \) in this subregion as the consequence of the rule:

\[
\hat{e}_i = \sum_{m=1}^{H} v_m L_m(\xi_{mi}) + \tau
\]

\[
\xi_{mi} = x'_iw_m
\]

2. Generate the rule condition: \((C_1 \text{ and } C_2 \text{ and } \cdots \text{ and } C_H)\), where \( C_m \) is either \( \xi_{mi} < -\xi_{m0} \) or \( -\xi_{m0} \leq \xi_{mi} \leq \xi_{m0} \) or \( \xi_{mi} > \xi_{m0} \).

Thus, a rule set which is equivalent to the pruned NN with 3-piece linear approximation can be represented in the following form:

\[
\xi_{m1}; \xi_{m2}; \ldots; \xi_{mH} \in \mathbb{R}, \text{ the intercepts of the 3-piece linear approximation of the } H \text{ hidden neurons which determine the regions.}
\]
Rule 1: IF Region 1 THEN $\hat{e}_i = E_1$
Rule 2: IF Region 2 THEN $\hat{e}_i = E_2$

Rule P: IF Region P THEN $\hat{e}_i = E_P$

with P the number of non empty regions and

\[
E_1 = \sum_{m=1}^{H} v_m L_m(x'_i, w_m) + \tau
\]

\[
E_2 = \sum_{m=1}^{H} v_m L_m(x'_i, w_m) + \tau
\]

\[
\vdots
\]

\[
E_P = \sum_{m=1}^{H} v_m L_m(x'_i, w_m) + \tau
\]

with $x_i$ the input samples that lies within the associated region.

4. EMPIRICAL SET-UP

In this section, background information concerning the data set used is given and the data preprocessing steps are detailed. The set-up of the study and the different evaluation criteria to assess the techniques are also discussed.

4.1 Data set

The ISBSG, International Software Benchmarking Standards Group, is a not-for-profit organization that maintains a large data set of software project data. This data set is often used by researchers (e.g. [20] [29] [38] [23]). In this study, ISBSG R11 (May 2009) is used containing 5052 projects collected from companies worldwide; Fig. 4 gives a breakdown of the origin of the projects. The data relates to projects collected from 1992 up until 2009.

Table 2 provides an overview of the parameters included in the data set. In case of categorical variables with more than 8 possible values, some of the levels were merged to obtain a more concise data set. This merging was based on semantic similarity. Levels with less than 15 observations, were put in a category named ‘Other’ in line with a study done by Jeffrey et al. [23]. The categorical variables were transformed into binary variables by applying dummy encoding as the categorical attributes are nominal by nature [5]. A missing data flag was created to account for missing values. No outlier removal or other preprocessing steps were applied.

4.2 Study set-up

To obtain a fair estimation of performance of the various techniques, the data is randomly split into a training and a test set containing 649 projects and 72 projects respectively.
The techniques are trained using only data contained in the training set and afterwards evaluated using the previously unseen data from the test set.

A key question of any estimation method is whether the predictions are accurate; the difference between the actual effort, \( e_i \), and the predicted effort, \( \hat{e}_i \), should be as small as possible. The two most common used criteria in the context of software effort prediction are MMRE and Pred(25). Both are derived from the Magnitude of Relative Error (MRE) [14]. The MRE is calculated for each observation and is defined as:

\[
MRE_i = \left| \frac{e_i - \hat{e}_i}{e_i} \right|
\]

The MMRE (Mean MRE) is defined as:

\[
\text{MMRE} = \frac{100}{N} \sum_{i=1}^{N} \left| \frac{e_i - \hat{e}_i}{e_i} \right|
\]

A complementary accuracy measure is \( \text{Pred}(L) \) [14], the fraction of observations for which the predicted effort, \( e_i \), falls within L% of the actual effort, \( \hat{e}_i \):

\[
\text{Pred}(L) = \frac{100}{N} \sum_{i=1}^{N} \left\{ \begin{array}{ll} 1 & \text{if } \frac{e_i - \hat{e}_i}{e_i} \leq \frac{L}{100} \\ 0 & \text{otherwise} \end{array} \right. 
\]

Typically, the \( \text{Pred}(25) \) measure is considered, looking at the percentage of predictions that are within 25% of the actuals.

While both measures are based on the MRE, they have a slightly different focus. \( \text{Pred}(L) \) is favoring models which are generally accurate but occasionally widely inaccurate. MMRE on the other hand can be highly affected by outliers [35]. To address this shortcoming in the MMRE measure, the MdMRE is also considered. The MdMRE is the median of all MREs and thus can be considered more robust against outliers.

\[
\text{MdMRE} = 100 \times \text{median}(\text{MRE})
\]

Finally, the techniques are also assessed using the \( R^2 \) measure. This measure reflects the percentage of variation that is being explained by the model:

\[
R^2 = 1 - \frac{SS_{err}}{SS_{tot}} = 1 - \frac{\sum_{i=1}^{N} (e_i - \hat{e}_i)^2}{\sum_{i=1}^{N} (e_i - \overline{e})^2}
\]

\( R^2 \) is a value between 0 and 1 with 1 meaning all the variation in the data is explained by the model.

The results obtained on the independent test set are tested using one way ANOVA (ANalysis Of Variance). This statistical technique assesses the question whether the mean across multiple groups are different by taking the variances of the groups into account, hence its name. The results are tested at statistical significance level \( \alpha = 0.01 \). Following the ANOVA test, Tukey’s honest significance test is utilized to perform a pairwise comparison of the techniques in order to investigate whether the differences between two specific techniques are statistically significant. A statistical significance level of \( \alpha = 0.05 \) is used for this second test.

5. RESULTS

In a first part of the experiment, the three techniques (OLS, CART and rule extraction from NNs) are compared to each other in terms of MMRE, MdMRE, Pred(25) and \( R^2 \). The results of the experiments are displayed in Table 3. It can be seen that CART provides the best overall results while the rule extraction algorithm is second except when the \( R^2 \) is considered. However, analysis of the CART tree shows that the resulting tree consists of 47 splitting nodes. The induced rule set thus consists of 48 rules with in some cases up to 10 rule antecedents. The CART tree can thus be considered too elaborate to be easily comprehensible by the business. The rule set induced from the pruned NN is considerably smaller, only containing 5 rules with at most 2 rule antecedents. The rule set obtained from the pruned NN is given in Table 4.

In a second part of the experiment, we further pruned the CART tree in order to obtain the same complexity of rule set (the same number of rules and rule antecedents) as the rule set extracted from the pruned NN. The results are indicated in Table 3 as ‘CART pruned’ and the CART tree as well as the induced rule set are shown in Fig. 1. It can be concluded from the results that the NN rules perform better than the pruned CART tree in terms of MMRE, MdMRE, Pred(25) and \( R^2 \). Fig. 5 shows box plots for the 4 techniques of the MMRE. The median value which is displayed in the box plots is the MdMRE which is also provided in Table 3.

To further assess the statistical significance of the results, an ANOVA is performed to test whether the MMRE is significantly different across the 4 techniques (OLS, CART, CART pruned and Rule extraction from NN). The null hypothesis is the following:

\[
H_0 : \text{MMRE}_{\text{CART}} = \text{MMRE}_{\text{OLS}} = \text{MMRE}_{\text{CART pruned}} = \text{MMRE}_{\text{Rrules}}
\]

The results of the ANOVA are given in Table 5. The test is significant at 1% (\( p=0.000149 \)) thus it can be concluded that the MMRE across the 4 techniques is statistically different. Following the ANOVA, a Tukey’s honest significance test is performed which does a pairwise comparison of the different techniques. The null hypothesis is given below.

\[
H_0 : \text{MMRE}_k = \text{MMRE}_l \text{ with } k \neq l
\]

The results of this test are displayed in Table 6. In bold-face font the significant pairwise differences are given (criti-
It can be concluded from these tests that the regression rules extracted from the NN are significantly better performing than the pruned CART tree (MMRE\textsubscript{CARTpruned} = 186.45 versus MMRE\textsubscript{OLS} = 95.39). The CART tree without additional pruning is the best performing technique at the expense of a larger and thus less comprehensible rule set although this result is only statistically significant when compared to the pruned CART tree. When CART is compared with OLS, the result is very close to the critical value (3.625 as compared to the critical value = 3.722).

### 6. CONCLUSIONS

This experimental study assessed the feasibility of regression rule extraction from Neural Networks (NN) in the context of software effort prediction. The algorithm generates a small number of linear equations from a neural network trained for regression. The analysis on the ISBSG R11 data set indicated that the performance of the rule set obtained by applying the REFANN (Rule Extraction from Function Approximating Neural Networks) algorithm on a trained and pruned NN is on par with other techniques. However, the regression rule extraction algorithm provides the end user with a more concise set of rules (both in terms of the number of rules and the number of rule antecedents) than the CART (Classification And Regression Tree) algorithm which is commonly used in the literature. Therefore the idea of regression rule extraction should be considered of high relevance to the practical implementation of software effort prediction systems in a business environment where comprehensibility is of high importance.

Further research on the usability of the algorithm on other software effort prediction data sets is needed to assess this technique more in depth. Other types of rule extraction algorithms are currently investigated by the authors.

### 7. ACKNOWLEDGMENTS

This research was supported by the Odysseus program (Flemish Government, FWO) under grant G.0915.09. We would also like to thank the Flemish Research Fund for financial support to the authors (Odysseus grant G.0915.09 to Bart Baesens and post-doctoral research grant to David Martens).

### References


